Fall 2014 **CMSC 351: Midterm** Hamid Mahini
Time: 1 Hour, 30 Minutes

Type A

WAIT FOR INSTRUCTIONS BEFORE BEGINNING

HONOR PLEDGE: "I pledge on my honor that I have not given or received any unauthorized assistance on this examination."

Signature and UID: _____

Print name: _____

- ***Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily. You should prove the correctness of your algorithms either directly or by referring to a proof in the book.***
- ***The sum of the grades is 105, but your grades would be out of 100 (thus you get 5 bonus points by solving all the problems).***
- ***Select the best choice for the first 5 problems and mark it by X in the table below***

Problem	1	2	3	4	5
A					
B					
C					
D					
E					

DO NOT WRITE BELOW THIS LINE

Problem 1-5:	
Problem 6:	
Problem 7:	
Problem 8:	
Problem 9:	
TOTAL:	

Multiple-choice Problems (Answer ONLY in the TABLE in the FIRST PAGE and NOT HERE):

Problem 1. (5 points) Let T(n)=7T($\lfloor \frac{n}{3} \rfloor$)+ n^2-4n and T(0)=1. Then T(n) is

a) $\Theta(n^2 \log n)$ b) $\Theta(n^2)$ c) $\Theta(nlogn)$
d) $\Theta(nlog^2 n)$ e) $\Theta(n^{log_3^7})$

Problem 2. (5 points) Consider hash table H of size 7 and hash function $h(x) = (2x + 4)\ mod\ 7$. If we use the linear probing method for resolving collisions, what would be H[0] after the following operations (execute them from left to right):

Insert(4), Insert(8), insert(5), insert(7), delete(5), insert(1), insert(11), insert(5)

a)1 b)5 c)7 d) 11 e) $null$

Problem 3. (5 points) The following procedure calculates 2^n for any natural number *n*. What is its running time?

```
power(n) {
  if (n == 0)
    return 1;
  else
    return power(⌊n/2⌋) × power(⌈n/2⌉);
}
```

a) $\Theta(1)$ b) $\Theta(log\ n)$ c) $\Theta(n)$ d) $\Theta(n\ log\ n)$ e) $\Theta(n^2)$

Problem 4. (5 points) The following procedure calculates 2^n for any natural number *n*. What is its running time?

```
power(n) {
  if (n == 0)
    return 1;
  a = power(⌊n/2⌋);
  if (n is odd)
    return 2 × a × a;
  else
    return a × a;
}
```

a) $\Theta(1)$ b) $\Theta(log\ n)$ c) $\Theta(n)$ d) $\Theta(n\ log\ n)$ e) $\Theta(n^2)$

Problem 5. (5 points) What would be A[4] if we remove the maximum element from the following heap?

Index	1	2	3	4	5	6	7	8	9	10
A	20	17	15	13	9	3	7	2	6	8

a) 6 b) 8 c) 9 d) 13 e) 17

Regular Problems:

Problem 6. (20 points) Consider the following functions:

(func 0) $\log(n)^2$
(func 1) $1.8^n - n$
(func 2) $\log(n^3)$
(func 3) $\sqrt{4n+6}$
(func 4) $\frac{n!}{2^n}$
(func 5) $200n^{0.01} + 3$

Complete the table below. In the entry at row "$func\ i$" and column "$func\ j$", cross the best relation you can prove between "$func\ i$" and "$func\ j$", i.e. put ONLY ONE of O or Ω or Θ. (You do not need to write the proofs).

Functions	$func\ 0$	$func\ 1$	$func\ 2$	$func\ 3$	$func\ 4$	$func\ 5$
$func\ 0$	Θ	O	Ω	O	O	O
$func\ 1$	Ω	Θ	Ω	Ω	O	Ω
$func\ 2$	O	O	Θ	O	O	O
$func\ 3$	Ω	O	Ω	Θ	O	Ω
$func\ 4$	Ω	Ω	Ω	Ω	Θ	Ω
$func\ 5$	Ω	O	Ω	O	O	Θ

Problem 7. (20 points) Prove by induction that $9^n - 1$ is divisible by 4 for every positive integer n.

Base case: we have $9^1 - 1 = 9 - 1 = 8 = 4 \times 2$. So the statement holds for n=1.

Induction Hypothesis: For every positive integer m<n, $9^m - 1$ is divisible by 4.

Inductive Step: We have to prove $9^n - 1$ is divisible by 4.

$$9^n - 1 = 9^n - 1 + 9 - 9 = 9^n - 9 + 8 = 9 \times (9^{n-1} - 1) + 8$$

We know $9^{n-1} - 1$is divisible by 4 based on the induction hypothesis, and thus we have $9^n - 1 = 4k$ for an integer k. Now we can rewrite $9^n - 1$ as:

$$9^n - 1 = 9 \times (9^{n-1} - 1) + 8 = 9 \times 4k + 8 = 4 \times (9k + 2)$$

This means $9^n - 1$ is divisible 4.

Problem 8. (20 points) Assume we have a binary search tree and each node stores the size of its subtree where the size of a subtree is the total number of nodes in that subtree. Design an algorithm which calculates the rank of a given number x in the binary search tree. The rank of number x is equal to the number of elements in the binary search tree that are less than or equal to x. Your algorithm should run in $O(h)$ where h is the height of the binary search tree. Describe your algorithm and write its pseudo-code.

Define variable rank which is initially 0. Start from the root and in each step compare x to the value of the current node. If x is equal to the value of the current node, just add the size of its left subtree+1 to variable rank and return rank. If x is less than the value of the current node, just go to the left subtree. If x is greater than the value of the current node, add the size its left subtree+1 to the rank and go to the right subtree. Here is the peudo-code:

```
int find-rank(Node* current, intx, int rank) {
        if (current.left != null)
            leftSize = current.left.size;
        else
            leftSize = 0;
        if (current.value == x)
              return rank + leftSize + 1;
        else if (current.value > x)
                return find-rank(current.left, x, rank);
            else
                return find-rank(current.right, x, rank+leftSize + 1)
}
```

Problem 9. (20 points) Consider the following modified version of mergesort algorithm. Partition the array into three equal parts. Run mergesort on each part and merge the three parts back together.

A. Design an $O(n)$ algorithm for merging the three sublists together.
B. Using the merge procedure above, design an algorithm for sorting in this modified version.
C. Give a recurrence and analyze the running time of your algorithm.

A. Assume we want to merge three sublists A1, A2, and A3. Define three pointers i1, i2, and i3 which are pointing to the current smallest elements of each sublist. At each step we find the smallest element among A1[i1], A2[i2], and A3[i3], add it to the result array, and increment the corresponding pointer. Note that in each step we would find the next element of the output by finding the smallest element among A1[i1], A2[i2], and A3[i3], and thus each step takes O(1). Since the total number of step is equal to the size of the output which is n, we can conclude that the running time of the proposed merge procedure is O(n).

B. Here would be the merge sort:

```
void mergesort(A, first, last) {
        if (first >= last)
            return A;
        mid1 = first + (last-first)/3;
        mid2 = first + 2*(last-first)/3;
        L  = mergesort(A, first, mid1);
        M = mergesort(A, mid1+1, mid2);
        R = mergesort(A, mid2+1, last);
        return merge(L, M, R);
}
```

C. Since procedure merge is O(n) and we call three sub-problem of size n/3, the recurrence relation would be the following:

$$T(n) = 3 * T(n/3) + O(n)$$

We can use the master theorem (with a=3, b=3, and c=1) and conclude that the running time is $O(nlog\ n)$